TP7 – Reconnaissance vocale

Dans le TP6, vous avez vu que l'information contenue dans un sonagramme pouvait être condensée sous la forme d'une empreinte sonore, et que c'est sur cette notion d'empreinte sonore que repose le principe de Shazam. Or, même s'il s'agit d'un problème proche, la reconnaissance vocale ne peut être envisagée à l'aide de l'empreinte sonore, car la variabilité de prononciation, l'accent, la hauteur et le timbre de voix du locuteur rendent la reconnaissance nettement plus compliquée. Le but de ce TP est de vous montrer comment on peut réaliser cette tâche.

Analyse cepstrale d'un signal acoustique

La transformée de Fourier discrète (TFD) d'une suite de N valeurs $x(0) \dots x(N-1)$ est la suite de N valeurs $X(0) \dots X(N-1)$ définies par :

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-2i\pi k \frac{n}{N}}, \qquad k = 0 \dots N-1 \tag{1}$$

En pratique, les N valeurs $x(n)$ sont les échantillons $x_n = x(nT_0)$ d'un signal analogique $x(t)$, et les N valeurs $X(k)$ forment une approximation de la transformée de Fourier de $x(t)$ pour les N fréquences $f_k = k\,\frac{1}{NT_0}$, $k = 0 \dots N-1$.

Le *cepstre* d'un signal acoustique $x(t)$, notion apparue au début des années 1960 (ceps = spec lu à l'envers), est égal à la partie réelle de la transformée de Fourier inverse du logarithme du module de sa transformée de Fourier. Par analogie avec un sonagramme, qui est obtenu par concaténation de spectres correspondant aux positions successives d'une fenêtre glissante (cf. TP5), on peut calculer un cepstre pour les positions successives d'une fenêtre glissante. La *fenêtre de Hamming* est une fonction positive, à support borné de largeur L, égale à $0,54 - 0,46\,\cos(2\pi\frac{t}{L})$ lorsque $t \in [0, L]$, et nulle sinon. La figure 1 montre la chaîne complète de calcul des coefficients cepstraux.

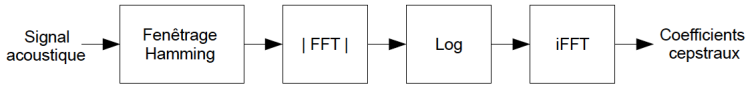

FIGURE 1 – Chaîne de calcul des coefficients cepstraux.

Exercice 1 : analyse cepstrale de la voix

Le répertoire `Voyelles` contient 90 fichiers sonores au format `WAV` correspondant à 9 phonèmes de type voyelle : /a/ (pl**a**t), /ə/ (l**e**), /e/ (bl**é**), /ɛ/ (col**è**re), /i/ (**i**l), /o/ (ch**au**d), /ɔ/ (m**o**rt), /u/ (gen**ou**), /y/ (r**u**e). Chaque phonème est prononcé 10 fois : 5 fois par une voix masculine, et 5 fois par une voix féminine. Pour une même voyelle, la hauteur et la durée du signal changent d'un enregistrement à l'autre. Quelle représentation de ces signaux sonores doit-on utiliser pour pouvoir distinguer les voyelles ? Le but de cet exercice est de vous montrer que le cepstre est le plus adapté, car il permet de séparer les contributions respectives de la source et du conduit vocal.

Faites une copie de la fonction `gabor` sous le nom `spec_ceps`. Modifiez cette fonction, qui est appelée par le script `exercice_1`, de manière à calculer les coefficients spectraux et cepstraux du signal passé en paramètre : les coefficients spectraux découlent des deux premières étapes de la figure 1. Utilisez la fonction `hamming` de Matlab, et choisissez comme décalage la demi-largeur de la fenêtre de Hamming. Sachant que le nombre de colonnes des paramètres `spec` et `ceps` retournés par cette fonction dépend de la durée de l'enregistrement, et que la prononciation d'une voyelle constitue un signal à peu près constant (contrairement à une consonne), le script `exercice_1` calcule le spectre moyen et le cepstre moyen. Le nuage de 90 points de $\mathbb{R}^{882}$ ainsi obtenu est ensuite partitionné en 9 classes correspondant aux différentes voyelles, à l'aide la méthode des k-moyennes (fonction `kmeans`). Manifestement, au vu des pourcentages de bonnes classifications obtenus, le cepstre est plus adapté à la reconnaissance vocale que le spectre. Il en va de même après réduction par ACP de la dimension des données, qui passe de 882 à 3.

Algorithme DTW

La principale difficulté de la reconnaissance vocale vient de ce que les locutions verbales ne correspondent généralement pas à des signaux constants dans le temps, mis à part pour les voyelles de l'exercice 1. Au lieu d'effectuer une classification dans $\mathbb{R}^{882}$ ou dans $\mathbb{R}^3$, on doit *aligner* la locution de test avec les locutions d'une base d'apprentissage, et déterminer la locution d'apprentissage la plus proche de la locution de test.

L'algorithme DTW (*Dynamic Time Warping*) est un algorithme générique qui permet d'aligner des séquences de n'importe quel type, par exemple des séquences d'ADN (exercice 2) ou des locutions verbales (exercice 3). Bien entendu, la fonction de distance entre un item de la première séquence et un item de la deuxième séquence dépend de l'application visée. Un avantage non négligeable de l'algorithme DTW est qu'il est très rapide, car il s'inspire du principe de la *programmation dynamique*.

Exerice 2 : alignement de séquences d'ADN

Le script `exercice_2` est censé aligner les deux séquences d'ADN suivantes :

```
AAGTAGGC
ATGGTACGTC
```

Écrivez la fonction `alignement`, qui doit permettre d'aligner deux séquences de n'importe quel type. Cette fonction, dont le principe a été vu en cours, reçoit en entrée le nom de la fonction de distance entre un item de la première séquence et un item de la deuxième séquence. Dans le cas des séquences d'ADN, la fonction `distance_ADN` retourne 1 si les deux lettres passées en paramètres sont différentes, et 0 sinon. La fonction `alignement` doit retourner la matrice `g` des pénalités cumulées, le score de l'alignement optimal, tel qu'il a été défini en cours, et les matrices de booléens `i_prec` et `j_prec` qui permettent de garder en mémoire le numéro de ligne et le numéro de colonne de la case précédente le long du chemin optimal : si `i_prec(i,j)` est égal à 1, alors la case précédente de `(i,j)` se trouve sur la ligne `i-1` ; sinon, elle se trouve sur la ligne `i` (idem pour la signification de `j_prec`). **Attention :** le passage du paramètre `distance` à la fonction `alignement` nécessite d'utiliser la fonction `feval` (lisez la documentation de cette fonction).

Exercice 3 : alignement de locutions verbales

Le répertoire `Corpus` contient un *corpus* d'enregistrements sonores correspondant à 6 locutions verbales prononcées par 13 locuteurs masculins : « arrête-toi », « avance », « droite », « fais un flip », « gauche », « recule ». Ces locutions sont des ordres censés diriger un drone. Or, il convient de savoir si le cepstre constitue une donnée suffisamment discriminante pour interpréter l'ordre correctement.

Écrivez la fonction `distance_locutions`, dont le nom est passé en paramètre lors de l'appel de la fonction `alignement` par le script `exercice_3`. Il s'agit tout simplement de la distance euclidienne dans $\mathbb{R}^n$, où n désigne le nombre de lignes des cepstres. Le script `exercice_3` considère le premier locuteur comme le *locuteur de référence*, et calcule la matrice de confusion sur la totalité des 13 locuteurs (son calcul est relativement long). Cherchez quel locuteur de référence permet de minimiser le nombre de confusions.

Exercice 4 : commande vocale

Écrivez la fonction `reconnaissance_vocale`, qui doit permettre au script `exercice_4` de reconnaître une locution tirée au hasard parmi les 78 enregistrements du répertoire `Corpus`, puis d'effectuer une action induite par la locution reconnue. Libre à vous d'imaginer d'autres actions que celles proposées !

Enfin, il vous est demandé de contribuer à construire un nouveau corpus, en vous enregistrant sur *smartphone*. Lisez le fichier `aide_enregistrement.pdf`, et enregistrez les mêmes locutions verbales que celles du corpus initial. Les voix masculines étant très différentes des voix féminines, il pourra être opportun d'améliorer le programme de commande vocale en utilisant un locuteur de référence pour chaque sexe.